NFT Auctions by Kaggle Team lululu233

Lu Tan (301270445) Benson Ou-yang (301277342)

November 8, 2021

Contents

1 Introduction **2**

2 Data Description **2**

3 Methods **3**

3.1 Data Pre-processing 3

3.2 Feature Engineering 3

3.3 Machine Learning 3

3.4 Deep Learning 4

4 Results **4**

5 Conclusion **4**

6 References **4**

7 Challenge Question **5**

7.1 Summary Statistics of NFT Auction Prices 5

7.2 Visualizations 5

7.3 Linear Regression 6

8 Bonus Question **7**

8.1 Alibi Detect 7

8.2 Outputs 8

1 Introduction

Non-fungible tokens (NFTs) are unique digital items that are bought and sold via cryptocurrency. The hype behind these NFTs is the insanely high amounts of Ethereum spent for them. Many celebrities and big brands have launched their own NFTs that many people are hoping to collect. The big question is what makes certain NFTs worth more than others. For our analysis, we gathered the NFT auction data and did some data cleaning. We also did some image and text processing for more features. Afterwards, we fit a linear model with stochastic gradient descent with different explanatory variables. By trying to improve the *Mean Absolute Error*, we compared some models with default parameters and ran a fivefold *Cross Validation* process. To further improve our analysis, we tried implementing a *Deep Neural Network* and a *Convolutional Neural Network*. With the *Keras* package, we were able to combine a *Multilayer Perceptron Model* with a *Convolutional Neural Network* to produce NFT auction price predictions.

2 Data Description

Table 1: Description of the features

Variable Name	Description
id	The NFT ID
X.sales	The number of times the NFT has been auctioned previously
cdate	The date when the NFT was created
description	A modified text description where each word is an unique string
fee1	Transaction fee associated with the auction
fee2	Transaction fee associated with the auction
total	Closing auction price

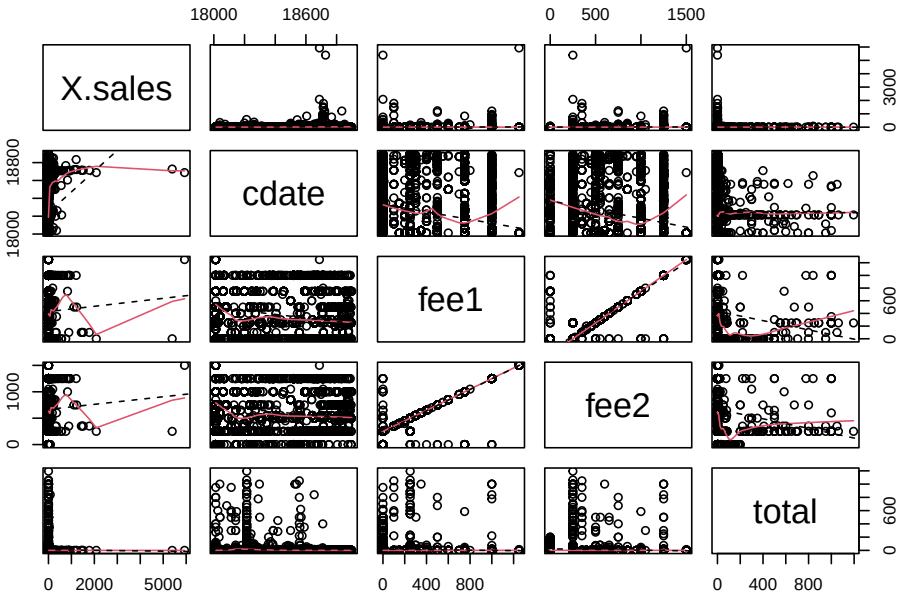

Figure 1: Pairs Plots of Explanatory Variables and Response Variable

From Figure 1, fee1 and fee2 could be potential categorical variables. X.sales have points in an L shape where many NFT auctions only have one sale, and a few NFTs have multiple.

3 Methods

3.1 Data Pre-processing

We first start with cleaning the data by filling all NAN values with 0. Next, we standardize the features to make the data consistent.

3.2 Feature Engineering

3.2.1 Image Processing

From the package *imageio*, we extracted features from the images that are in the training set. We denoted each column as fi1 to fi7. The columns represented the width, height, brightness, darkness, and the mean intensity of red, green and blue.

3.2.2 Text Processing

3.2.2.1 Text Feature Extraction

We first converted the text to a matrix of token count with *CountVectorizer* from the package *scikit-learn*. Next, we removed the top ten percent of the most frequent words to avoid common stop words. Afterwards, we transform the matrix into a corpus for further analysis.

3.2.2.2 Text Modelling

Latent Dirichlet Allocation(LDA) is a three-level Bayesian model, with each item is modelled as a mixture of topics. It classifies text data based on some assumptions, such as independent topics and exchangeable words.

Non-negative Matrix Factorization(NMF) is an effective method of clustering high-dimensional data. It reduces the dimension by separating the term-by-document matrix into two-dimensional factor matrices. Each vector in the original matrix is a bag-of-words representation for every document.

Truncated Singular Value Decomposition(Truncated SVD) is a process that linearly reduces the number of input variables known as dimensionality reduction. By reducing the dimensionality, this method can preserve the meaningful features observed in the data.

Kernel Principal Component Analysis{KPCA} is a nonlinear method that reduces dimensions through the use of kernels. It helps with decision boundaries of data that are nonlinear.

3.3 Machine Learning

The first model we attempted was fitting a linear regression model by minimizing a regularized empirical loss with *Stochastic Gradient Descent*. In the package *scikit-learn*, we used the function *SGDRegressor*. The hyperparameters we chose were epsilon insensitive for the loss function, zero for both the alpha and epsilon. With the loss function as epsilon insensitive, the model ignores errors less than epsilon. Alpha is the constant that multiplies the regularization term.

We also tried fitting many regressor models from *scikit-learn* such as *Lasso, ElasticNet, KernelRidge, GradientBoostingRegressor, XGBRegressor, and LGBMRegressor*. By using a five-fold *Cross Validation* process and splitting the training data into a train and test set, we were able to compare these models by calculating the *Mean Absolute Error*(MAE). By averaging the MAE, we discovered that the *Gradient Boosting Regressor* performed the best. We then fine-tuned the hyperparameters by using *GridSearchCV*.

3.4 Deep Learning

By extracting the grayscale image data into an array, we fit a *Deep Neural Network* and a *Convolutional Neural Network* from the package *Keras*. With the NFT auction data, we fit a *Multilayer Perceptron* model. To further improve our analysis, we concatenated the *Convolutional Neural Network*(CNN) with a *Multilayer Perceptron*(MLP). The MLP took the numerical data as input while the CNN took the image arrays as inputs, and the output was the NFT auction price predictions.

4 Results

The text processing methods with the *SGD Regressor* that improved the MAE were NMF and Truncated-SVD. *SGDRegressor* was the best performing model with the explanatory variables X.sales, cdate, fee1,fee2 and the text processing features. The *Gradient Boosting Regressor* improved quite a bit after tuning the hyperparameters but did not beat the linear model. Our best model was the combination of the *Convolutional Neural Network*(CNN) and the *Multilayer Perceptron*(MLP).

5 Conclusion

In our analysis, we discovered the importance of feature selection as that affects the strength of the models. A way to further minimize the *Mean Absolute Error* is to have better feature engineering. Although we did some feature engineering, our scores did not improve drastically. Furthermore, if we researched a little more about the *Neural Networks* and the *Deep Learning* model process and parameter tuning, it could potentially increase the ability to predict. In conclusion, using the combination of *CNN* and *MLP* model was the most successful at predicting NFT auction prices.

6 References

Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow, 2019

Pedregosa et al., Scikit-learn: Machine Learning in Python, 2011

Abadi et al., TensorFlow: Large-scale machine learning on heterogeneous systems, 2015

Chollet et al., Keras, 2015

Tolios, Simplifying Image Outlier Detection with Alibi Detect, 2020

7 Challenge Question

7.1 Summary Statistics of NFT Auction Prices

Table 2: Summary Statistics

	Total
count	6914.000
mean	9.660
std	73.850
min	0.000
25%	0.029
50%	0.120
75%	0.450
max	1195.000

7.2 Visualizations

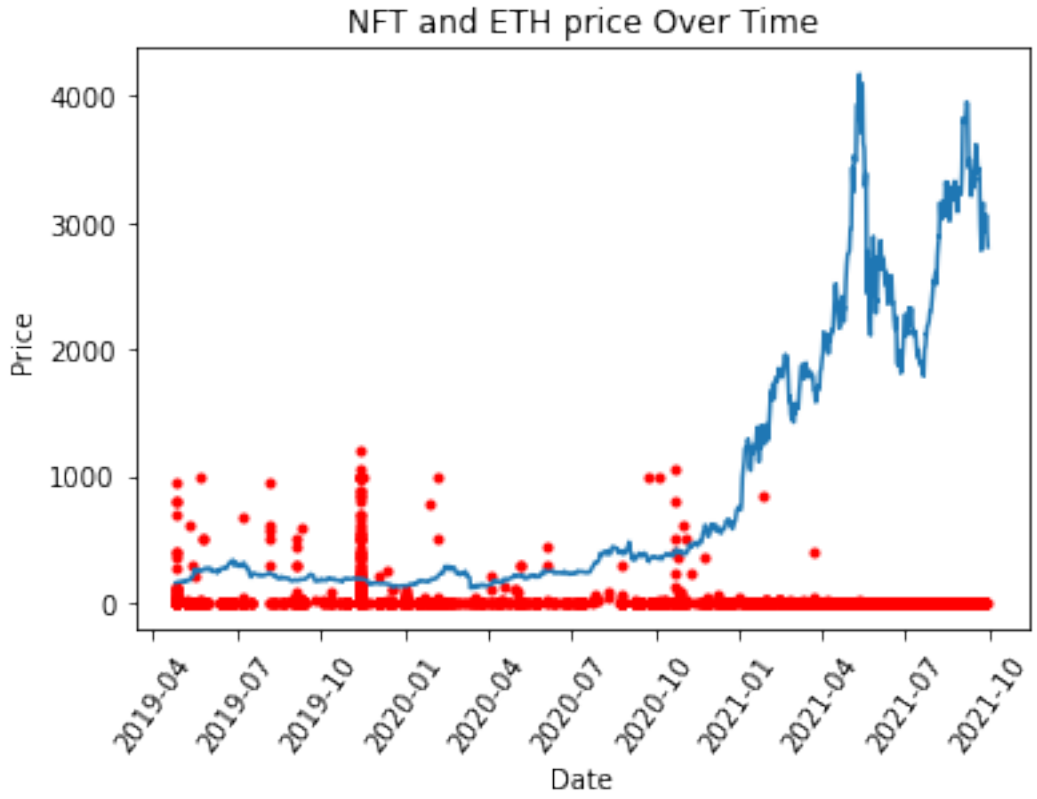

Figure 2: Comparison between ETH and NFT prices over time

From Figure 2, there seem to be a lot of data points at the bottom that is worth investigating. From the summary statistics table, the mean is 9.66, and the median is 0.12. Many outliers are driving the mean NFT price up, so removing the outliers or just studying the data points around the median may be beneficial.

7.3 Linear Regression

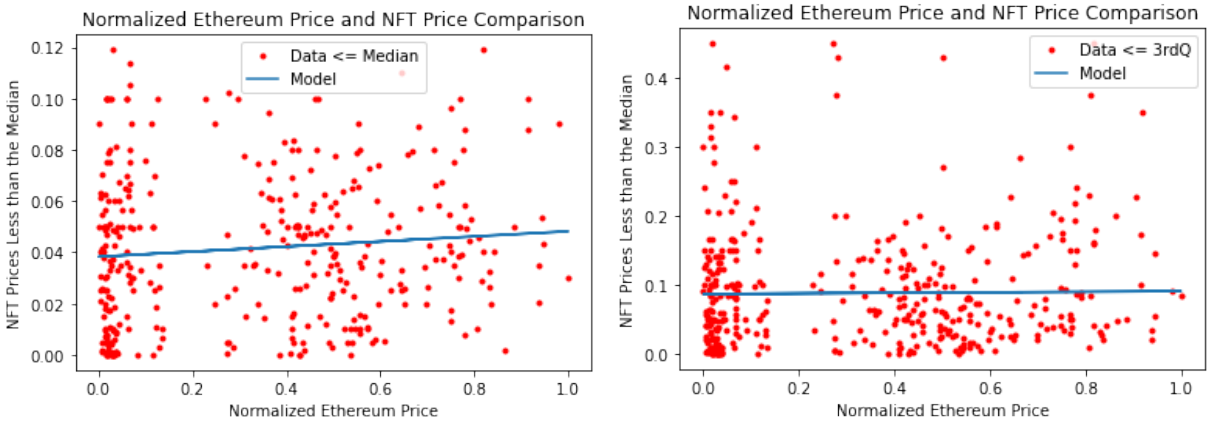

Figure 3: Median & 3rd Quartile NFT Auction Prices and Normalized ETH prices with Regression Line

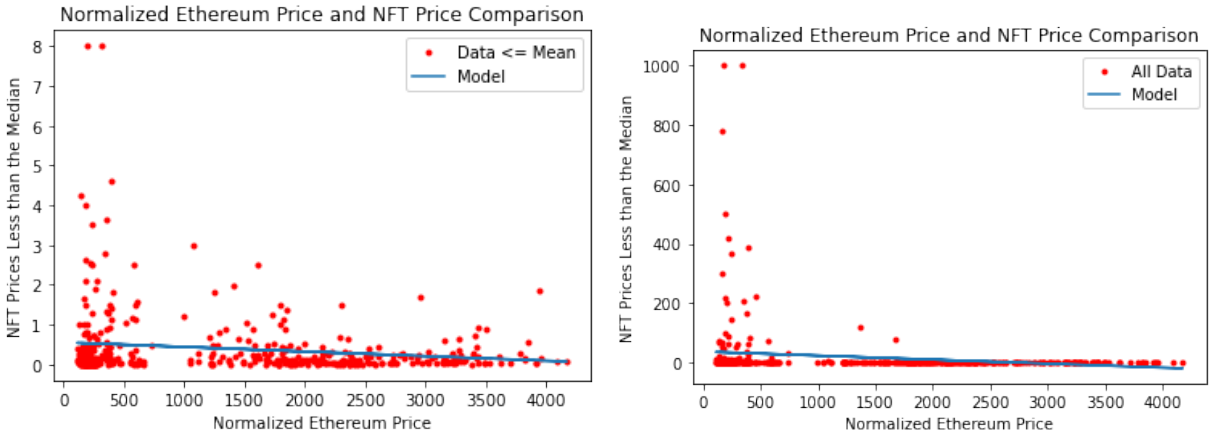

Figure 4: Mean & All NFT Auction Prices and Normalized ETH prices with Regression Line

From the above plots (Figures 3, 4), we ran a linear regression model to see the correlation between Ethereum price and the NFT sale price. We normalized the Ethereum price to make the plots easier to interpret. There seems to be a slightly positive correlation between Ethereums price and the NFT sale price when comparing only the NFT prices that are less than the median. We also made models based on all NFT sale prices, prices less than the third quartile and the mean. There is a positive effect between Ethereum prices and NFT auction prices less than the median sale prices. As we include more data, the correlation starts to decrease.

8 Bonus Question

8.1 Alibi Detect

```
# inspired by
# https://towardsdatascience.com/simplifing-image-outlier-detection-with-alibi-detect-6aea686bf7ba

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import glob
import os
import shutil
from collections import Counter
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Conv2DTranspose, UpSampling2D,\
    Dense, Layer, Reshape, InputLayer, Flatten, Input, MaxPooling2D
from alibi_detect.od import OutlierAE
from alibi_detect.utils.visualize import plot_instance_score, plot_feature_outlier_image

def img_to_np(te1, resize = True):
    img_array = []
    for ii in range(te1.shape[0]):
        if ii % 100 == 0:
            print('%d / %d' % (ii, te1.shape[0]))
        if te1['ext'][ii] == '.png':
            id = te1.loc[ii,'id']
            ff = te1.loc[ii, 'id'] + te1.loc[ii, 'ext']
            path = 'data/images/images/' + ff
            if not os.path.isfile(path):
                continue

            img = Image.open(path).convert("RGB")
            if(resize):
                img = img.resize((64,64))
            img_array.append(np.asarray(img))
    images = np.array(img_array)
    return images



#path_train = "data/images/images"
#path_test = "data/images/images"

train = img_to_np(te1 = tr1)
test = img_to_np(te1 = te1)
train = train.astype('float32') / 255.
test = test.astype('float32') / 255.
```

This code chunk imports all required libraries. The function *img_to_np* takes a file path to an image and

converts it into a NumPy array. We used the functions twice to make the training and testing data for the model.

```
encoding_dim = 1024
dense_dim = [8, 8, 128]

encoder_net = tf.keras.Sequential(
  [
      tf.keras.layers.InputLayer(input_shape=train[0].shape),
      tf.keras.layers.Conv2D(64, 4, strides=2, padding='same', activation=tf.nn.relu),
      tf.keras.layers.Conv2D(128, 4, strides=2, padding='same', activation=tf.nn.relu),
      tf.keras.layers.Conv2D(512, 4, strides=2, padding='same', activation=tf.nn.relu),
      tf.keras.layers.Flatten(),
      tf.keras.layers.Dense(encoding_dim,)
  ])

decoder_net = tf.keras.Sequential(
  [
      tf.keras.layers.InputLayer(input_shape=(encoding_dim,)),
      tf.keras.layers.Dense(np.prod(dense_dim)),
      tf.keras.layers.Reshape(target_shape=dense_dim),
      tf.keras.layers.Conv2DTranspose(256, 4, strides=2, padding='same', activation=tf.nn.relu),
      tf.keras.layers.Conv2DTranspose(64, 4, strides=2, padding='same', activation=tf.nn.relu),
      tf.keras.layers.Conv2DTranspose(3, 4, strides=2, padding='same', activation='sigmoid')
  ])

od = OutlierAE( threshold = 0.001,
                encoder_net=encoder_net,
                decoder_net=decoder_net)

adam = tf.keras.optimizers.Adam(lr=1e-4)

od.fit(train, epochs=100, verbose=True,
       optimizer = adam)

od.infer_threshold(test, threshold_perc=95)

preds = od.predict(test, outlier_type='instance',
            return_instance_score=True,
            return_feature_score=True)
```

This chunk contains the code for the encoder and decoder part of the *Convolutional Autoencoder*. The *Convolutional Autoencoder* is a form of *Convolutional Neural Networks*. They reconstruct images while minimizing errors by learning optimal filters. After this training process, they apply to any inputs for feature extraction.

This model calculates a threshold value where any outputs instance score above this value is an outlier.

8.2 Outputs

```
k=0
a = np.zeros(len(preds['data']['is_outlier']))
```

```
for ii in range(len(preds['data']['is_outlier'])):
    if te1['ext'][ii] == '.png':
        id = te1.loc[ii,'id']
        ff = te1.loc[ii, 'id'] + te1.loc[ii, 'ext']
        path = 'data/images/images/' + ff
        if(preds['data']['is_outlier'][ii] == 1):
            source = path
            a[k] = ii
        k = k+1


dict1 = {'Filename': te1['id'][a],
     'instance_score': preds['data']['instance_score'],
     'is_outlier': preds['data']['is_outlier']}

df = pd.DataFrame(dict1)
df_outliers = df[df['is_outlier'] == 1]

print(df_outliers)
```

```
                              Filename  instance_score  is_outlier
0.0  7e79f1a9cb10504dd2fc569d84f2a346        0.010793           1
0.0  7e79f1a9cb10504dd2fc569d84f2a346        0.007903           1
0.0  7e79f1a9cb10504dd2fc569d84f2a346        0.013233           1
0.0  7e79f1a9cb10504dd2fc569d84f2a346        0.007496           1
0.0  7e79f1a9cb10504dd2fc569d84f2a346        0.010613           1
..                                  ...             ...         ...
0.0  7e79f1a9cb10504dd2fc569d84f2a346        0.008793           1
0.0  7e79f1a9cb10504dd2fc569d84f2a346        0.006753           1
0.0  7e79f1a9cb10504dd2fc569d84f2a346        0.012693           1
0.0  7e79f1a9cb10504dd2fc569d84f2a346        0.009666           1
0.0  7e79f1a9cb10504dd2fc569d84f2a346        0.007574           1

[168 rows x 3 columns]
```

Figure 5: DataFrame containing information about the outliers

Figure 5 shows the DataFrame containing the outliers information from the model prediction. It includes the filename(id), instance score and if it is an outlier.

```
print(df_outliers['Filename'].unique())
```

```
['7e79f1a9cb10504dd2fc569d84f2a346' 'd5949754359d4e95c5cbaf274a4efc47'
 '31565d91136605f5496c9347634c5a20' '59fcc23207ab2b1827b9f869f6f37286'
 '19426c7d6e7b17fde2a0c0a39d119c6c' '15dd1bf9d5843fa7edf3980a567cc5df'
 '2162bee0ef18ede6b5d328fcaaf571c2' '0814865104e83918db0fc73b306c9dbe'
 '497e0c8c3d9c59cbd748441ce8f3848d' '8e6b843b8ee10ae458e9fa7c2af21e01']
```

Figure 6: Unique NFT id that are outliers

Figure 6 outputs the unique ids that are outliers.

```
print(df_outliers.sort_values(by = ['instance_score']))
```

	Filename	instance_score	is_outlier
0.0	7e79f1a9cb10504dd2fc569d84f2a346	0.006053	1
0.0	7e79f1a9cb10504dd2fc569d84f2a346	0.006055	1
0.0	7e79f1a9cb10504dd2fc569d84f2a346	0.006056	1
0.0	7e79f1a9cb10504dd2fc569d84f2a346	0.006102	1
0.0	7e79f1a9cb10504dd2fc569d84f2a346	0.006128	1
..	...	...	...
0.0	7e79f1a9cb10504dd2fc569d84f2a346	0.016813	1
0.0	7e79f1a9cb10504dd2fc569d84f2a346	0.017598	1
0.0	7e79f1a9cb10504dd2fc569d84f2a346	0.017924	1
0.0	7e79f1a9cb10504dd2fc569d84f2a346	0.018191	1
0.0	7e79f1a9cb10504dd2fc569d84f2a346	0.020931	1

[168 rows x 3 columns]

Figure 7: Sorted instance scores of outliers

Figure 7 is the sorted DataFrame of the outlier predictions. The top and bottom five rows are the ids of the same image.

```
print(df_outliers.explode('Filename')['Filename'].value_counts())
```

```
7e79f1a9cb10504dd2fc569d84f2a346    159
d5949754359d4e95c5cbaf274a4efc47      1
31565d91136605f5496c9347634c5a20      1
59fcc23207ab2b1827b9f869f6f37286      1
19426c7d6e7b17fde2a0c0a39d119c6c      1
15dd1bf9d5843fa7edf3980a567cc5df      1
2162bee0ef18ede6b5d328fcaaf571c2      1
0814865104e83918db0fc73b306c9dbe      1
497e0c8c3d9c59cbd748441ce8f3848d      1
8e6b843b8ee10ae458e9fa7c2af21e01      1
Name: Filename, dtype: int64
```

Figure 8: Frequency of NFT outliers detected

Figure 8 shows that only one image got detected multiple times.

```
outimg = 'data/images/images/7e79f1a9cb10504dd2fc569d84f2a346.png'
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread(outimg)
imgplot = plt.imshow(img)
plt.show()
```

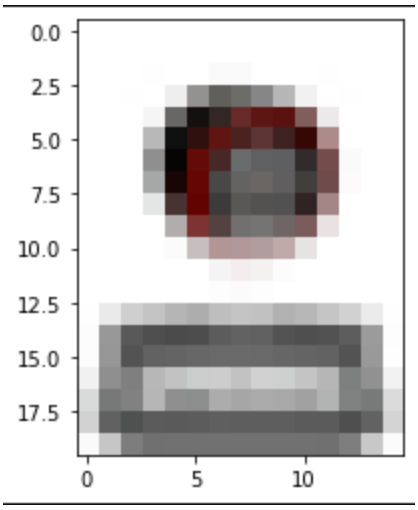

Figure 9: Outlier image out of the NFT thumbnails

Figure 9 is the image detected as an outlier many times. Although the picture is scaled down, the shape and colours potentially indicate a child's drawing to the human eye. This image id is 7e79f1a9cb10504dd2fc569d84f2a346.